NorESM code analysis and optimization

Final report

SNIC 2014/1-155 and SNIC 2014/8-18

1. Project organization

Requester Name: Ilona Riipinen
Title/position: Assoc. Professor
Affiliation: Department of Applied Environmental Science, Stockholms University
E-mail: ilona.riipinen@itm.su.se

Project responsible for requester
Name: Juan-Camilo Acosta
Title/position: PhD student
Affiliation: Department of Applied Environmental Science, Stockholms University
E-mail: juan-camilo.acosta@itm.su.se

Project responsible for SNIC
Name: Chandan Basu
Title/position: Application expert
Affiliation: NSC, SNIC
E-mail: cbasu@nsc.liu.se

Project member
Name: Hamish Struthers
Title/position: Application expert
Affiliation: NSC, SNIC
E-mail: struthers@nsc.liu.se

Project manager
Name: Torben Rasmussen
Title/position: Application expert
Affiliation: NSC, SNIC
E-mail: torbenr@nsc.liu.se

2. Project objective

The project aims to understand the performance of the NorESM code, which is a global Earth system model. The main objectives are:

- Review the MPI pe mapping of the different sub-models (atmosphere, land, sea ice and ocean) currently used for model simulations.
- Analyze the NorESM code to identify code sections and routines to be further evaluated for performance optimization.
- Propose a set of recommendations for code optimization changes based on the performance analysis.

3. Resource utilization

The project was conducted during 2014-H2. NSC has spent total 1 PM within this project. The work was started in the middle of September, 2014 and continued till December 2014. We used the SNIC 2014/1-155 and SNIC 2014/8-18 resource allocations on Triolith as the compute resource for the project.

4. Project Report

4.1 Load balancing of NorESM runs

The NorESM scaling / performance is dependent on the load balancing of different components. We have used N20TRAERCNTRCH test case for our analysis. In NorESM Land (LND), Ice (ICE) and Ocean (OCN) models run concurrently. Also Atmosphere (ATM) and Ocean (OCN) models run concurrently. However LND and ICE models run sequentially with ATM model. The process layout of our test runs are shown in *Figures 1 – 3*.

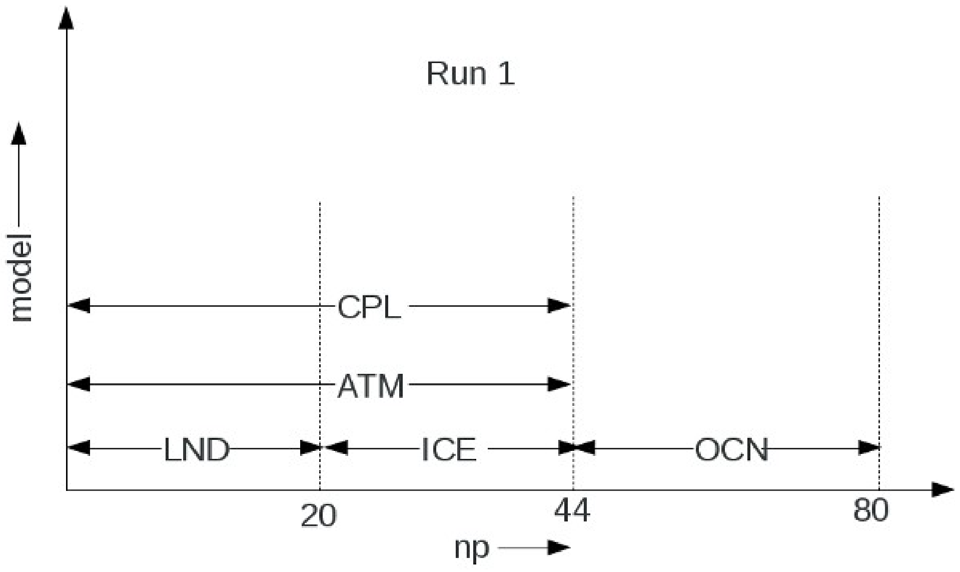


Figure 1

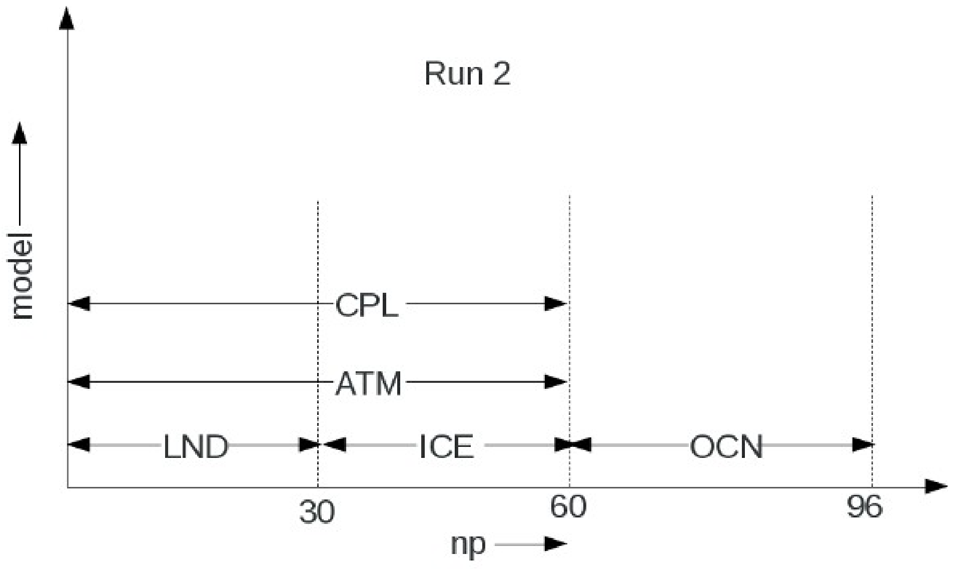


Figure 2

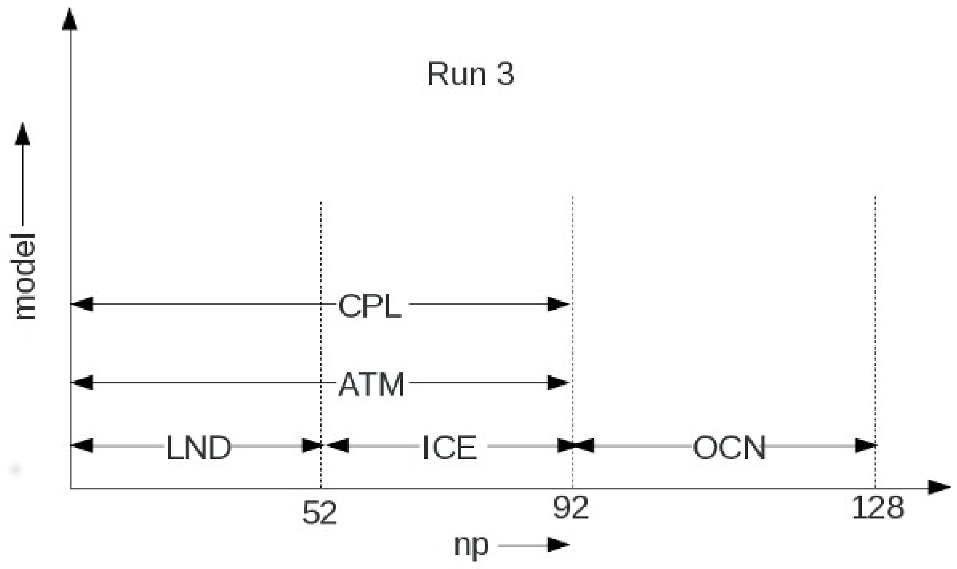


Figure 3

In *Table 1* below we show the run times for the three different tests

tests	t(s)	np
Run1	632	80
Run2	544	96
Run3	425	128

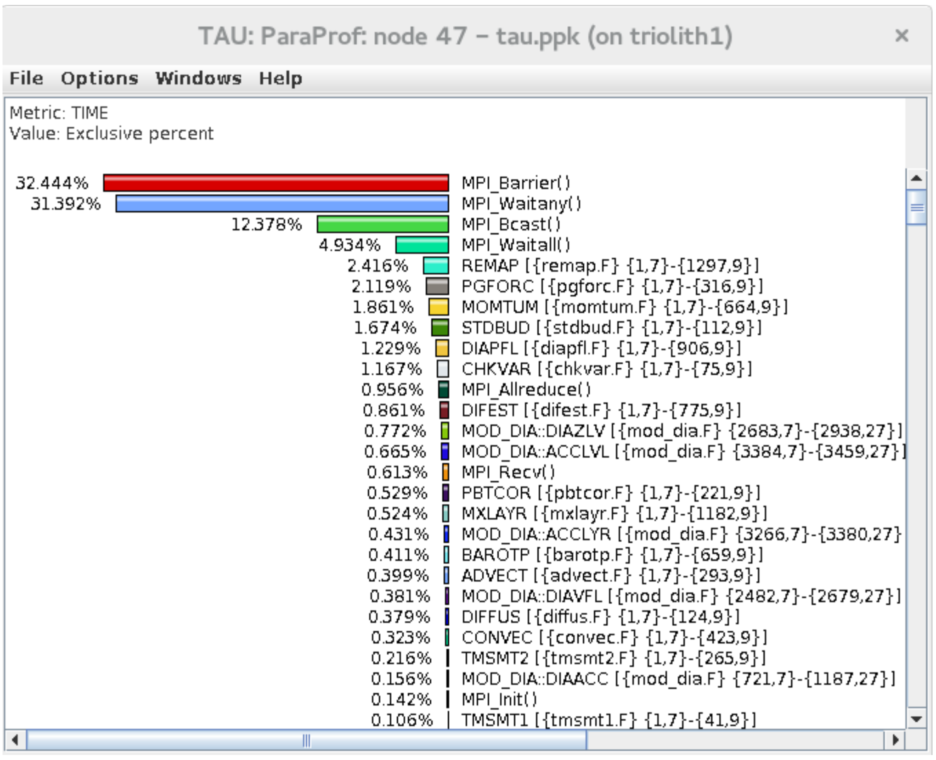
Table 1

For optimized runs the ATM and OCN components should be perfectly load balanced. Also LND and ICE components should be load balanced. In our tests OCN model is run on 36 processes. The ATM model is run on 44, 60 and 92 processes respectively for Run1, Run2 and Run3 tests. As the overall run time reduces with increasing no. of ATM processes we can conclude that in all these three test cases OCN component was not the bottleneck and the ATM part was load imbalanced compared to the OCN component. We have done TAU based profiling to see the load imbalance of different components.

4.2 Ananlysis of the NorESM run with TAU

We have run NorESM with TAU profiler. For this we first compile the code with TAU compiler wrapper. When the code is run the TAU profiler library records the call time of MPI routines and the user routines. The resulting profiles can be viewed using paraprof visualizer. In *Figure 4* we show profile for a typical OCN process for our Run1 test case. As can be seen 32% and 31% of times are spent respectively on MPI_Barrier and MPI_Waitany routines. This unusually high MPI time may be due to the fact that the OCN processes are waiting for ATM processes. In *Figure 5* we show profile for a typical OCN process for Run3 test case. It can be seen that 26% and 15% of times are spent respectively in MPI_Barrier and MPI_Waitany routines. Although the total no. of

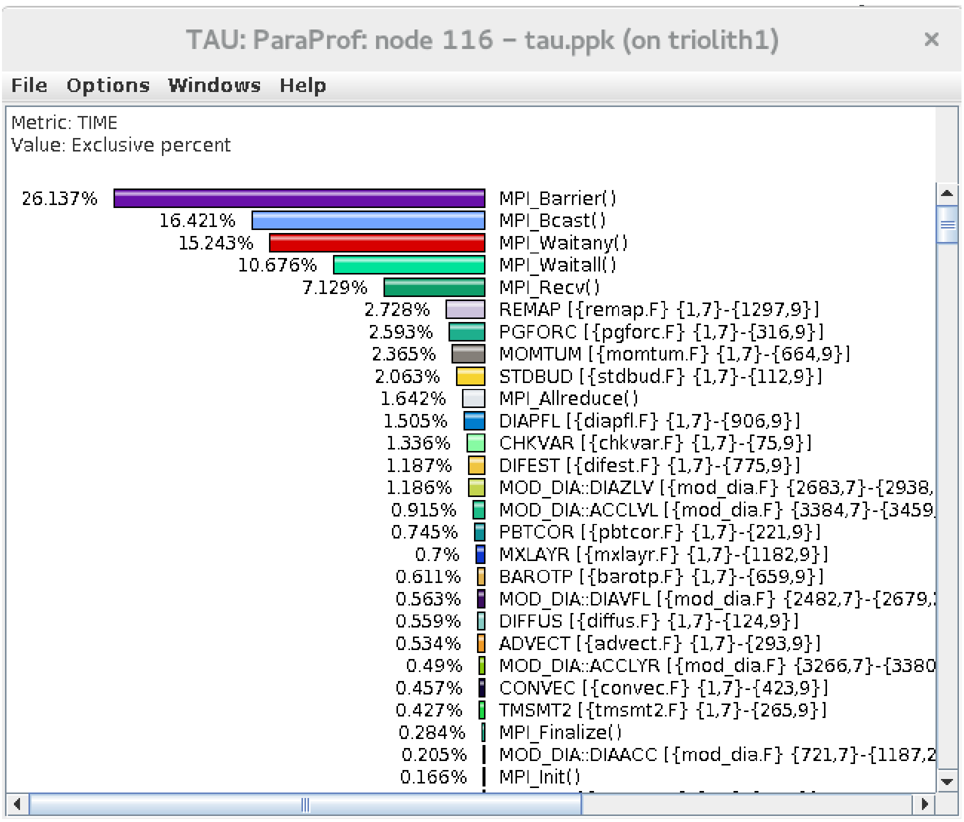
OCN processes have remained same between Run1 and Run3 there is reduction in MPI_Barrier and MPI_Waitany time. The total MPI time for Run1 and Run3 tests are 85% and 75% respectively. This improvement is due to the fact that the OCN processes have less waiting for the ATM processes. It also seems that the very high MPI time is also due to the fact that for our test case OCN model is run on too many no. of processes than is optimal.


TAU: ParaProf: node 47 – tau.ppk (on triolith1)
File Options Windows Help

```
Metric: TIME
Value: Exclusive percent

32.444%  MPI_Barrier()
31.392%  MPI_Waitany()
12.378%  MPI_Bcast()
4.934%   MPI_Waitall()
2.416%   REMAP [{remap.F} {1,7}-{1297,9}]
2.119%   PGFORC [{pgforc.F} {1,7}-{316,9}]
1.861%   MOMTUM [{momtum.F} {1,7}-{664,9}]
1.674%   STDBUD [{stdbud.F} {1,7}-{112,9}]
1.229%   DIAPFL [{diapfl.F} {1,7}-{906,9}]
1.167%   CHKVAR [{chkvar.F} {1,7}-{75,9}]
0.956%   MPI_Allreduce()
0.861%   DIFEST [{difest.F} {1,7}-{775,9}]
0.772%   MOD_DIA::DIAZLV [{mod_dia.F} {2683,7}-{2938,27}]
0.665%   MOD_DIA::ACCLVL [{mod_dia.F} {3384,7}-{3459,27}]
0.613%   MPI_Recv()
0.529%   PBTCOR [{pbtcor.F} {1,7}-{221,9}]
0.524%   MXLAYR [{mxlayr.F} {1,7}-{1182,9}]
0.431%   MOD_DIA::ACCLYR [{mod_dia.F} {3266,7}-{3380,27}
0.411%   BAROTP [{barotp.F} {1,7}-{659,9}]
0.399%   ADVECT [{advect.F} {1,7}-{293,9}]
0.381%   MOD_DIA::DIAVFL [{mod_dia.F} {2482,7}-{2679,27}]
0.379%   DIFFUS [{diffus.F} {1,7}-{124,9}]
0.323%   CONVEC [{convec.F} {1,7}-{423,9}]
0.216%   TMSMT2 [{tmsmt2.F} {1,7}-{265,9}]
0.156%   MOD_DIA::DIAACC [{mod_dia.F} {721,7}-{1187,27}]
0.142%   MPI_Init()
0.106%   TMSMT1 [{tmsmt1.F} {1,7}-{41,9}]
```

Figure 4: A typical OCN process profile for Run1 case

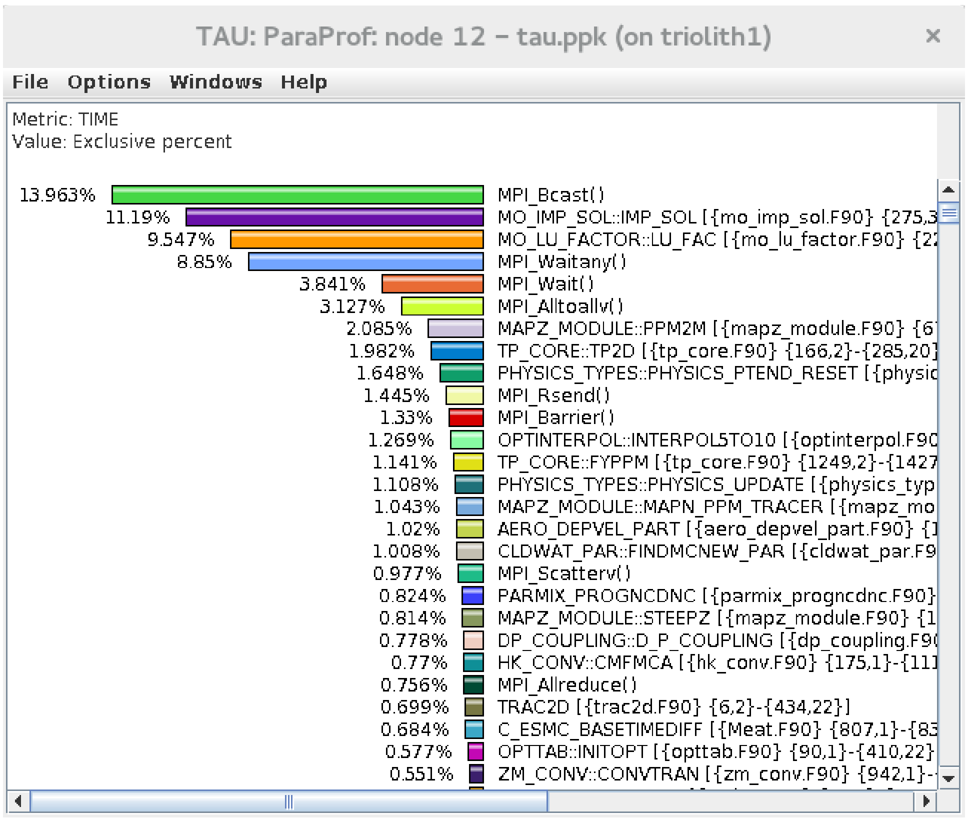

TAU: ParaProf: node 116 – tau.ppk (on triolith1)
File Options Windows Help

```
Metric: TIME
Value: Exclusive percent

26.137%  MPI_Barrier()
16.421%  MPI_Bcast()
15.243%  MPI_Waitany()
10.676%  MPI_Waitall()
7.129%   MPI_Recv()
2.728%   REMAP [{remap.F} {1,7}-{1297,9}]
2.593%   PGFORC [{pgforc.F} {1,7}-{316,9}]
2.365%   MOMTUM [{momtum.F} {1,7}-{664,9}]
2.063%   STDBUD [{stdbud.F} {1,7}-{112,9}]
1.642%   MPI_Allreduce()
1.505%   DIAPFL [{diapfl.F} {1,7}-{906,9}]
1.336%   CHKVAR [{chkvar.F} {1,7}-{75,9}]
1.187%   DIFEST [{difest.F} {1,7}-{775,9}]
1.186%   MOD_DIA::DIAZLV [{mod_dia.F} {2683,7}-{2938,
0.915%   MOD_DIA::ACCLVL [{mod_dia.F} {3384,7}-{3459,
0.745%   PBTCOR [{pbtcor.F} {1,7}-{221,9}]
0.7%     MXLAYR [{mxlayr.F} {1,7}-{1182,9}]
0.611%   BAROTP [{barotp.F} {1,7}-{659,9}]
0.563%   MOD_DIA::DIAVFL [{mod_dia.F} {2482,7}-{2679,
0.559%   DIFFUS [{diffus.F} {1,7}-{124,9}]
0.534%   ADVECT [{advect.F} {1,7}-{293,9}]
0.49%    MOD_DIA::ACCLYR [{mod_dia.F} {3266,7}-{3380
0.457%   CONVEC [{convec.F} {1,7}-{423,9}]
0.427%   TMSMT2 [{tmsmt2.F} {1,7}-{265,9}]
0.284%   MPI_Finalize()
0.205%   MOD_DIA::DIAACC [{mod_dia.F} {721,7}-{1187,2
0.166%   MPI_Init()
```

Figure 5: A typical OCN process profile for Run3 case

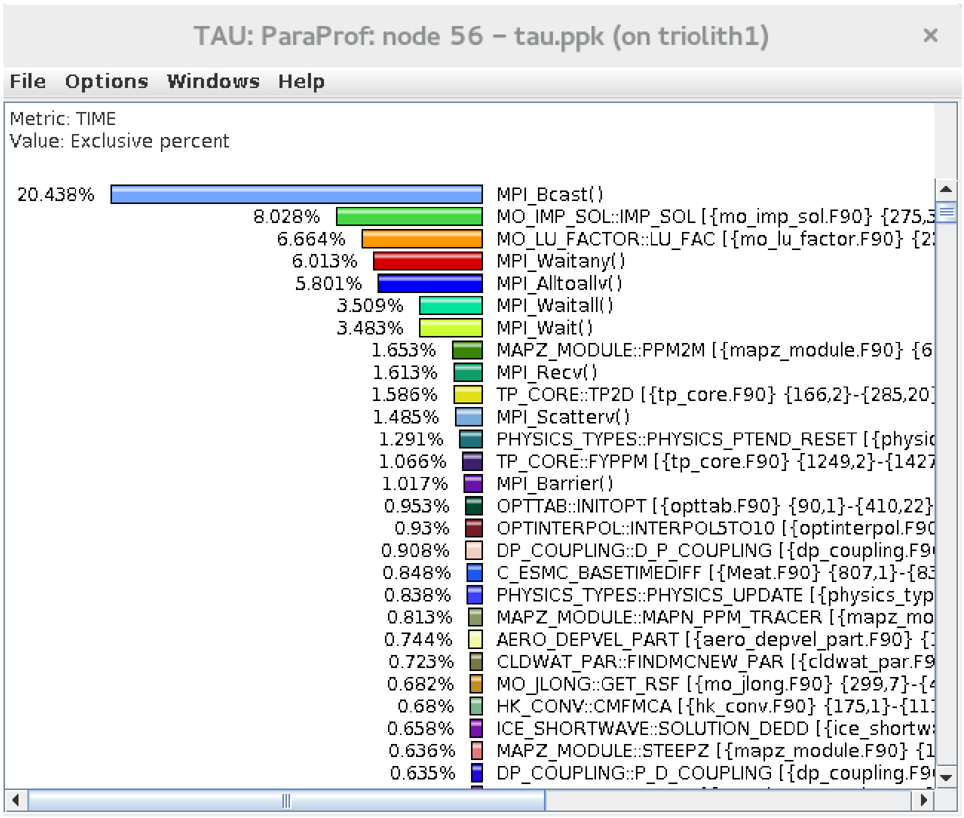
In *Figure 6* and *7* we show the profile of typical a ATM process for Run1 and Run2 tests respectively. For Run1 and Run2 respectively 14% and 20% of times are spent in MPI_Bcast routine. The increase in MPI_Bcast time is expected with increase in no. of ATM processes from Run1 to Run3. It can be seen from *Figures 6* and *7* that total MPI times are 35% and 45% respectively for Run1 and Run3. Looking at the total MPI time it seems that for this test case ATM model has reached it's scaling limit. If we look at the non MPI routine profiles (exclusive time) in *Figures 6* and *7* it shows that IMP_SOL and LU_FAC each take between 6 – 12 % of total run time. All other routines have small contributions to total run time. Hence it is obvious that optimization of these routines will have very little impact on total run time. Optimizing IMP_SOL and LU_FAC on the other hand might improve the run time. However there seems to be no low hanging fruits for optimization in these routines. It seems in LU_FAC it may be possible to use optimized LU factorization libraries. But it will require clear understanding of the data structures used by LU_FAC routines which is beyond the scope of this project.


TAU: ParaProf: node 12 – tau.ppk (on triolith1)
File Options Windows Help

```
Metric: TIME
Value: Exclusive percent

13.963%  MPI_Bcast()
 11.19%  MO_IMP_SOL::IMP_SOL [{mo_imp_sol.F90} {275,3
 9.547%  MO_LU_FACTOR::LU_FAC [{mo_lu_factor.F90} {2
  8.85%  MPI_Waitany()
 3.841%  MPI_Wait()
 3.127%  MPI_Alltoallv()
 2.085%  MAPZ_MODULE::PPM2M [{mapz_module.F90} {6
 1.982%  TP_CORE::TP2D [{tp_core.F90} {166,2}-{285,20}
 1.648%  PHYSICS_TYPES::PHYSICS_PTEND_RESET [{physic
 1.445%  MPI_Rsend()
  1.33%  MPI_Barrier()
 1.269%  OPTINTERPOL::INTERPOL5TO10 [{optinterpol.F90
 1.141%  TP_CORE::FYPPM [{tp_core.F90} {1249,2}-{1427
 1.108%  PHYSICS_TYPES::PHYSICS_UPDATE [{physics_typ
 1.043%  MAPZ_MODULE::MAPN_PPM_TRACER [{mapz_mo
  1.02%  AERO_DEPVEL_PART [{aero_depvel_part.F90} {1
 1.008%  CLDWAT_PAR::FINDMCNEW_PAR [{cldwat_par.F9
 0.977%  MPI_Scatterv()
 0.824%  PARMIX_PROGNCDNC [{parmix_progncdnc.F90}
 0.814%  MAPZ_MODULE::STEEPZ [{mapz_module.F90} {1
 0.778%  DP_COUPLING::D_P_COUPLING [{dp_coupling.F9
  0.77%  HK_CONV::CMFMCA [{hk_conv.F90} {175,1}-{11
 0.756%  MPI_Allreduce()
 0.699%  TRAC2D [{trac2d.F90} {6,2}-{434,22}]
 0.684%  C_ESMC_BASETIMEDIFF [{Meat.F90} {807,1}-{83
 0.577%  OPTTAB::INITOPT [{opttab.F90} {90,1}-{410,22}
 0.551%  ZM_CONV::CONVTRAN [{zm_conv.F90} {942,1}-
```

Figure 6: A typical ATM process profile for Run1 case


TAU: ParaProf: node 56 – tau.ppk (on triolith1)
File Options Windows Help

```
Metric: TIME
Value: Exclusive percent

20.438%  MPI_Bcast()
 8.028%  MO_IMP_SOL::IMP_SOL [{mo_imp_sol.F90} {275,3
 6.664%  MO_LU_FACTOR::LU_FAC [{mo_lu_factor.F90} {2
 6.013%  MPI_Waitany()
 5.801%  MPI_Alltoallv()
 3.509%  MPI_Waitall()
 3.483%  MPI_Wait()
 1.653%  MAPZ_MODULE::PPM2M [{mapz_module.F90} {6
 1.613%  MPI_Recv()
 1.586%  TP_CORE::TP2D [{tp_core.F90} {166,2}-{285,20}
 1.485%  MPI_Scatterv()
 1.291%  PHYSICS_TYPES::PHYSICS_PTEND_RESET [{physic
 1.066%  TP_CORE::FYPPM [{tp_core.F90} {1249,2}-{1427
 1.017%  MPI_Barrier()
 0.953%  OPTTAB::INITOPT [{opttab.F90} {90,1}-{410,22}
  0.93%  OPTINTERPOL::INTERPOL5TO10 [{optinterpol.F90
 0.908%  DP_COUPLING::D_P_COUPLING [{dp_coupling.F9
 0.848%  C_ESMC_BASETIMEDIFF [{Meat.F90} {807,1}-{83
 0.838%  PHYSICS_TYPES::PHYSICS_UPDATE [{physics_typ
 0.813%  MAPZ_MODULE::MAPN_PPM_TRACER [{mapz_mo
 0.744%  AERO_DEPVEL_PART [{aero_depvel_part.F90} {1
 0.723%  CLDWAT_PAR::FINDMCNEW_PAR [{cldwat_par.F9
 0.682%  MO_JLONG::GET_RSF [{mo_jlong.F90} {299,7}-{4
  0.68%  HK_CONV::CMFMCA [{hk_conv.F90} {175,1}-{11
 0.658%  ICE_SHORTWAVE::SOLUTION_DEDD [{ice_shortw
 0.636%  MAPZ_MODULE::STEEPZ [{mapz_module.F90} {1
 0.635%  DP_COUPLING::P_D_COUPLING [{dp_coupling.F9
```

Figure 7: A typical ATM process profile for Run3 case

5. Conclusion

We have analyzed the The NorESM code for parallel efficiency. Main improvement in NorESM run time can come for the load balancing of the OCN and ATM components. For achieving this one should start with "small no." of ATM and OCN processes. The "small no." can be guessed from some previous run. One should do few short runs (few time steps, typically 5 to 10 minutes duration) increasing processor count of one component at a time and starting with ATM component first. This way one can reach optimal no. of processor count for both OCN and ATM components. We think it is possible to write some kind of a script which can automate this test process.

There are some user defined non MPI routines, e.g., IMP_SOL and LU_FAC which contribute significantly to the run time. However improving these routines seems not a easy task and will require deep understanding of the code.

During the execution of this project we have seen that configuring, building and running NorESM is not very smooth. NorESM is built on top of CESM with few changes. While NorESM uses most of CESM build and run scripts, the user needs to manually copy some files or need to edit some files manually for being able to build / run NorESM properly. In this project we have developed few scripts which makes it easier. This scripts can be made available to the users.